1 *(1 punto) Indique en qué consiste la variante de la política de coherencia de cachés mediante invalidación conocida como Read_Broadcast y en qué medida puede reducir el número de fallos por invalidación. Explique hasta qué punto sería útil en una situación en que existieran múltiples escritores y lectores accediendo a un mismo bloque.*

SOLUCIÓN

Si aplica una política de invalidación "pura", el acceso a cada una de las copias de un mismo bloque ha sido modificado por un procesador producirá un fallo (los llamados fallos por invalidación o *invalidation misses*). Con la variante *Read_Broadcast* se reduce a uno el número de este tipo de fallo por cada invalidación que se realice, puesto que se aprovecha la lectura del bloque actualizado por parte del primer procesador que se lo encuentre en estado no válido para actualizar las demás copias.

Esta variante de la política de invalidación resulta ideal en el caso de bloques en los que se acceda por un solo escritor y varios lectores, y se aleja de ese ideal conforme aumenta el número de procesadores que escriben en él, llegando al extremos donde todos los procesadores escriben en un patrón temporal de grano fino a no representar ninguna mejora con respecto a la política de invalidación pura.

2 *(1 punto) Indique las ventajas y defectos de las políticas de invalidación y actualización para mantener la coherencia de cachés. ¿Qué política cree que se comportaría mejor en el caso del acceso a un bloque que contuviese un cerrojo al que se accede en espera activa o spin_lock?*

SOLUCIÓN

En el caso de invalidación, en general, se disminuye el tráfico extra necesario para mantener la coherencia, y resulta ideal en una compartición secuencial o de grano gordo en la que durante un intervalo grande de tiempo sólo accede un procesador a un determinado bloque. Sin embargo, el número de fallos de cache se acumulará al existente en un comportamiento monoprocesador y que corresponde a la no presencia de un determinado bloque. Estos fallos acumulados darán lugar a que se demande una copia válida del bloque que ha producido el fallo y, por tanto, también añadirán tráfico en pro de la coherencia en el sistema. Dependiendo del patrón de acceso al bloque, este tráfico extra puede llegar a superar al que se genera en la política de actualización, en la que siempre que un bloque esté compartido se difunde su modificación para que el resto de copias se mantengan coherentes.

La ventaja fundamental de esta segudna política es que no se producen fallos añadidos debidos a la política de coherencia y que, en principio, todas las copias son siempre válidas, por lo que una secuencia de accesos en que diferentes procesadores accediesen en instantes próximos en el tiempo al mismo bloque (compartición que hemos llamado de grano fino) se comportaría de manera más adecuada que la invalidación, donde cada acceso produciría un fallo y la correspondiente demanda del bloque (este efecto se aliviaría con la variante que aparece en la pregunta anterior).

El caso de acceso a un cerrojo en espera activa, si ésta se realiza en su versión más rudimentaria con un test&set que constantemente consulta y modifica el bloque correspondiente (tal y como aparece en el enunciado de la pregunta siguiente) y con un patrón de acceso en que varios procesadores compitan pro ganar el acceso, parece más adecuado el uso de la actualización, puesto que de lo contrario cada procesador, en cada "vuelta" al spin del cerrojo, produciría fallo de cache y la consiguiente demanda del bloque modificado, que ya se hubiera obtenido en el caso de emplear actualización.

3 *(1 punto) Indique las diferencias entre un MP UMA, NUMA y un sistema tipo cluster.*

SOLUCIÓN

Los dos primeros son máquinas con una visión de memoria compartida. En el caso UMA la memoria está también físicamente compartida y el tiempo de acceso es siempre el mismo. El sistema de intercomunicación suele ser un simples bus, pero tanto éste como la propia memoria se convierten rápidamente en un cuello de botella y permite un número reducido de procesadores (se dice que es poco "escalable"), que en el caso de un bus suele rondar la treintena, aunque con estructuras más sofisticadas aunque poco habituales se ha llegado a

pasar de cien.

En el caso NUMA, la visión sigue siendo de memoria compartida, pero está físicamente distribuida entre los procesadores. En este caso, cada procesador tiene un "envoltorio" Hw que consigue la visión de memoria compartida, auque al final el acceso a direcciones que no están almacenadas localmente se transforme en paso de mensajes por un sistema de intercomunicación heredado de las máquina de memoria distribuida y siempre más sofisticado que un bus, como son los hipercubos, mallas, toros, etc. Estas máquinas disfrutan de la ventaja de la "escalabildiad" de que carecían las UMAs, y llegan a alcanzar más de dos mil procesadores, como es el caso del Cray T3E.

Los clusters son sistemas de computación que unen con una red, normalmente de altas prestaciones, un conjunto de máquinas que son, cada una, de por sí autónomas o que podráin funcionar solas. Estas máquinas pueden ser desde simples PCs de bajo coso hasta, a su vez, MP como los que acabamos de describir. Una capa de Sw porporciona una visión de máquina única, con todos los aspectos que aparecen en la llamada SSI, *Single System Image*. Es la tecnología más barata en el mundo del HPC y la que ha acabado imponiéndose en este camopo,

4 *(2,5 puntos) Programe la versión paralela con OpenMP para máquinas de memoria compartida del siguiente programa (véase adjunto add.c). Tenga en cuenta que debe compilar también en su versión secuencial.*

SOLUCIÓN

Para realizar la versión paralela de este programa en memoria compartida con OpenMP basta con añadir antes del bucler `for` la siguiente directiva de compilación:

```
#pragma omp parallel for shared(vector) reduction(+:result)
```

Optionalmente se puede indicar algún tipo de planificación (`static`, `dynamic`, `guided`) mediante `schedule` así como el `chunk` de dicha planificación.

5 *(2 puntos) El siguiente fragmento de código no es óptimo y provoca más invalidaciones y más tráfico del necesario ¿Por qué? Reescríbalo de forma más óptima.*

```
Test:  test&set  .R1, /lock
       bnz test
       ret
```

SOLUCIÓN

Si se analiza el código se observa que es una implementación de un cerrojo con espera activa. En la estructura que se plantea, cada procesador tendrá copia del blqoue que contiene la dirección del cerrojo y, consecuentemente, la política de invalidación hará que a cada vuelta al bucle de espera sean invalidadas las copias de los demás procesadores, con los ulteriores fallos de cache por invalidación.

Una posible manera de atenuar esta situación una estructura conocida como test&test&set donde primero se consulta (y no se modifica) el valor del cerrojo hasta encontrarlo "abierto" (supuesta una política de coherencia el valor que se lea será el correcto, aunque produzca un fallo por invalidación) y luego tratar de ejecutar el test&set, que si que modificaría el bloque y daría lugar a las invalidaciones pertinentes. Nótese que el número de procesadores que hubiesen pasado el "filtro" de la lectura pudiera ser grande y entonces entonces en el test&set se podrái estar en una situación similar a la original, aunque no tiene porque darse este caso normalmente. El pseudocódigo sería como sigue:

```
A:  while (LOAD(lock) = 1) do
nothing;
     if (TEST&SET(lock) = 0)
{
```

```
región crítica;
}
     else goto A;
```

Otra posibilidades son los reintentos con retardo o *backoff*, que experimentalmente, y en particular en el caso de un retardo progresivo exponencial, exhiben el mejor rendimiento.

6 *(2,5 puntos) Un procesador "i" en un sistema UMA en que se emplea un mecanismo de coherencia de cachés con implementación snoopy y protocolo MESI accede en lectura a su copia local en caché de la variable "A" cuyo correspondiente bloque se encuentra en estado invalidado (I). Suponiendo que dicho bloque está en la caché del procesador "j" en estado modificado (M), indique la secuencia de pasos que se da en el sistema para obtener el valor actualizado de "A". Señale qué modificaciones se introducen en el sistema.*

A continuación el procesador "i" desea escribir un nuevo valor en "A". Indique la secuencia de acciones a que da lugar esta escritura y las modificaciones que conlleva en el sistema.

SOLUCIÓN

Al tener el dato invalidado lo tiene que pedir. La cache del procesador "j" lo proporciona y se actualiza de paso la memoria principal, las dos caches marcan el bloque como `compartido`.

Al escribir sobre dicho dato manda una invalidación a las demás caches, lo que provoca que la cache del procesador "j" lo marque como `invalidado` y luego lo modifica, dejandolo en el estado `modificado`

```
/* Programa  ADD.C */
#include <stdio.h>
#include <stdlib.h>

#define N_ELEM 90000000

int main ()
{
    int i;
    double result=0.0;
    double *vector;

    vector = (double *)malloc(sizeof(double) * N_ELEM);
    vector [1] = 1;
    vector [2] = 2;
    vector [3] = 3;
    vector [10000001] = 10;
    vector [10000002] = 20;
    vector [10000003] = 30;
    vector [20000001] = 100;
    vector [20000002] = 200;
    vector [20000003] = 300;

    /* To be parallelized */
    for (i=0; i < N_ELEM; i++)  {
        result += vector[i];
    }

    printf("The addition of all the elements is %f\n", result);

    return(0);
}
```